Formatting Structured Documents: Batch versus Interactive?

Giovanni Coray, Rolf Ingold, Christine Vanoirbeek
Swiss Federal Institute of Technology at Lausanne

ABSTRACT
The use of a multi-task system seems to open up new perspectives in document preparation. This paper presents such an approach, bringing together the wide possibilities of old markup techniques with the convenience of recently appeared interactive systems. It requires a very clear separation between a document's content and its formatting specification. Furthermore the latter can be favourably expressed with a descriptive formalism based on the document's logical structure.

Introduction

The subject matter of this paper stems from ideas developed in the context of a research contribution made in Lausanne on a document preparation project. The initial goal to produce technical reports has been broadened to solve more general document preparation problems (flexibility, modularity).

As interactive editing systems that include sophisticated typographical features become more fashionable, one might expect traditional formatting techniques to give way. The fact that this is not really the case is due to the advantages and shortcomings inherent in either approach: fast viewing and nice man-machine interface on the WYSIWYG systems, highest typographical quality and greater portability of documents through a variety of text processing software on the markup based textfile formatters.

Attempting to combine the good sides of both above mentioned approaches entails several requirements. First, the formatting process needs a flexible parametrisation that provides descriptive formatting specification, clearly separated from the document's content. This approach should offer more flexibility and guarantee portability of a

document to several systems with different printing devices. Second, a multi-tasking environment should permit to blend user-comfort with the high typographic quality realized by sophisticated formatting functions.

The aim of this paper is not to describe all the problems that occur in text processing techniques in general. It takes into account features related to document producing, i.e. preparation for printing on paper. It presents neither storage managment and access methods to data-base nor utilities like spelling or syntax-checker.

The first chapter presents current approaches in text processing techniques and situates our new point of view, and its special requirements. Chapter two describes an architecture of a new document processing system which should conform to these requirements. A formalism based on the logical structure of a document to describe its formatting, is presented in chapter 3. Finally chapter 4 gives a short overview of the user's possibilities.

1. The formatting process

This chapter summarizes the state of the art in present day text processing developments, and contrasts our approach with other existing avenues.

1.1. State of the art

Present day text processing systems for document production can coarsely be classified into two families: interactive versus batch systems. We shall first characterize both families.

1.1.1. Batch-mode text processing

Historically, for obvious technological reasons such as the availability of alphanumeric terminals and the prohibitive size of existing formatting software, most text processing tools were used in batch mode. The text to be processed was held on printable files, containing both characters as actual text, and commands as layout conventions. Compilation of the whole file was the standard means to produce a readable document, when correctly printed (figure 1).

The merit of these large and somewhat clumsy systems is to provide the user with a wide range of formatting, layout and macro primitives. The specific needs of any document composition application can thus be met by adapting or combining the functions from a body of existing software.

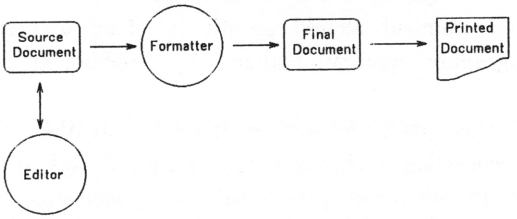

Figure 1: Architecture of batch formatter

Several drawbacks that would particularly weigh on the user's comfort and efficiency are inherent in this approach. The user is forced to pollute his text with quantities of commands, often spelled in illegible and rigid syntax; turn-around time for testing the commands is often too long for an efficient interaction during document "debugging", which as a whole appears as a fastidious and time consuming process.

A few systems palliate these drawbacks by simulating the printed document on a graphic screen [Chamberlin81], or by preprocessing the file for checkout purposes.

1.1.2. Interactive text processing

Recent technology, using graphic or bitmap screens, fast microprocessors, and less expensive laser printers, has given raise to an entirely new software. Individual workstations with fast graphic ouptut now are responsive to windows and mouse driven dialog.

Through the screen, the user has a realistic perception of his document as if it were printed (the ideal is coined WYSIWYG: What You See Is What You Get). This interaction is obtained through an integrated software that combines both editing and formatting (figure 2).

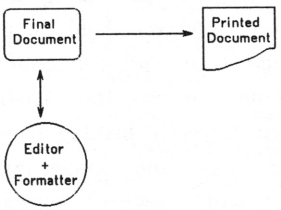

Figure 2: Architecture of interactive formatter

Although such systems are convincing by their user-friendly assets, such as fast response time, pointing devices to document sections

(mouse or stick), a.o.m., they suffer from a lack of flexibility on the functional level. They may provide a fair number of formatting commands, but at the cost of less elaborate structuring capabilities that would make them fit for more sophisticated applications. As it comes to extensions of their functionality these systems, by their very interactiveness, tend to shift the burden onto the user!

1.2. Objectives for a novel approach

The above considerations have led us to attempt a well integrated approach, that would combine the advantages of the old offline formatting technique, with the new practise of workstation type interaction. Let us detail the suggested objectives that such a combined approach ought to meet (figure 3).

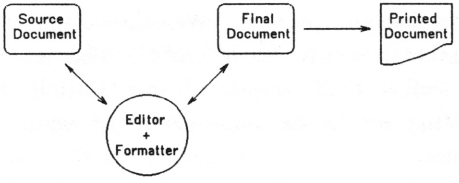

Figure 3: Integrated approach

1.2.1. Integration and extension: modularity

Besides the actual text, which constitutes the heart of the document, we usually find other entities such as logotypes, mathematical formulae, pretty- printed examples of Pascal programs or figures. In general, such graphical entities are produced by ad-hoc software [Lesk78]. From the formatting point of view they constitute inserts, either of binary data or of structured objects [VanDam82], within the final document.

Our preference goes to a system where such extraneous entities can be dealt with, not by tricks and patches, but according to the same general rules of an integrated concept. Any extension to the minimal set of formatting primitives ought to interface properly and explicitely with the others.

1.2.2. Parameterization

The last section gives room for vast generality. However, this precious potentiality should not burden the user with immoderate

complexity, unless directly required for his own application. We advocate a solution that allows the system to be tailored according to given user applications.

For instance, business correspondence sets requirements which are different, -almost complementary- to those expected in the process of producing scientific papers. One promising avenue is the a priori specification of the document's structure, which is then to serve as a guide for the editing and formatting process. We have chosen this type of parameterization of the process: by a document description. An adequate formalism for the structural description of a document will be presented in chapter 3.

1.2.3. Portability

Transferring a document from one textprocessing system to another remains a largely unsolved problem. Welcome standardisation efforts have been undertaken recently [Goldfarb81]. The actual trend is to use a descriptive rather than procedural specification, i.e. to abstract from any formatting action or implementation while presenting the document's structure.

Following this convincing philosophy we enforce a strict separation between the content of a document (i.e. the actual text and information to delimit its logical constituents) and its description (how this content is to be mapped onto typographical and layout conventions, for printing)

2. Architecture

This chapter presents a special architecture for a document production software which should satisfy closely the requirements set in chapter 1. After a brief presentation of the later used vocabulary one section analyses the formatting process, whereas another one sketches the system environment required.

2.1. Document structure and stepwise formatting

Different points of view may prevail when documents are produced. As a consequence, at least two types of structure can be associated with each document for formatting purposes:

- the logical structure, which corresponds to the natural decomposition of the matter as seen by the author;
- the typographical structure, which embodies all external aspects of the printed document, in particular its page and line breaks, its

layout, fonts a.s.o.

In our approach, at every time, we consider a document in several different forms, of which we retain the following two extreme aspects:

- the source document: it contains information given by the author, i.e. the textual data and its particular logical structure;
- the final document: it corresponds to the printed view of the document; it embodies all the typographical structure.

In this workframe, formatting can be defined as processing from source document to final document, stepping through one or more intermediate forms as indicated below.

2.2 Splitting of the formatting process

The conventional way of formatting in batch systems consists of running the whole source document to generate automatically the final document. This technique requires at first sight a response time proportional to the document length. This solution is therefore inadequate in an interactive context and all the more penalizing as the involved algorithms are sophisticated.

With the aim of increasing user convenience, it means conferring an interactive characteristic to the system, we split the formatting process of a whole document into several more elementary processes. The advantage of such a decomposition is to quickly provide the user with a partially and only locally formatted document that he can interact with.

On one hand, formatting can be split chronologically into several steps. For instance, line breaking of paragraphs can be done previously to page layout. On the other hand, documents including objects of same nature can be formatted independently from one another; among others this is the case for page breaking of the different chapters in a book.

Although this general scheme is quite appropriate, some particular situations are difficult to handle. This is the case when several apparently independant steps must satisfy interlaced constraints; a typical rule in good typography requires that words do not get split at a page break. Here some arbitration may be necessary, possibly by human interaction.

The interactive editing and the look-out of the different formatting steps naturally lead to the use of a multi-windows system where each one displays a piece of a partially formatted document. Its presence,

size and position on the graphic screen are driven by the user.

As an example, let us consider the following situation: a first window is associated with the source document such as it appears before any formatting operation; a second one shows a local portion of broken paragraphs; at last, a third one displays the aspect of the final document including page breaking and numbering.

Manipulation of document objects in the way described above is advantageously done in a multi-tasking environment of which a model is given in the next section.

2.3. Multi-tasking environment

Thus we consider on one hand an editing process to dialogue with the end-user and, on the other hand a number of formatting processes. All these processes are implemented as concurrent tasks under the control of a scheduler. The user is thus relieved from the obligation to direct explicitly all formatting steps.

The scheduler performs the processor's allocation according to a time-sharing scheme using fixed time slices. Furthermore it provides the necessary primitives to handle competition (mutual exclusion) and cooperation (synchronisation) of the coexisting tasks.

The data is shared by the processes in order to avoid redundant copies of the information. It is therefore necessary to provide mutual exclusion primitives for the access to the data by different processes.

Although the formatting algorithms can conveniently be expressed independently of each other, the order of their execution must obey strong constraints. For instance the process of formatting a chapter cannot be started unless all the paragraphs that occur within that chapter have previously been broken into lines. A synchronisation mechanism between the various processes is needed.

Last but not least an essential feature of our system is the dynamic nature of the formatting processes. Typically when a paragraph is modified due to an editing command, while its formatting is under way, the process associated with it must be killed, and subsequently replaced by a new process started on the modified data.

The system environment for a multistep formatting procedure must evidently support time-sharing and dynamic parallel processes.

3. The formal description of a document type

This chapter exhibits a formalism to describe first the logical structure of a source document and second the way it must be formatted.

3.1. Structure driven formatting of a document

According to our model, a document to be composed for typesetting should have two parts: the textual content and a style description valid for that type of document. Document samples that are destined for the same publication, such as papers that fall under the same heading in a periodical, will vary in content, but their style will be identical. Conversely, the same text may be subject to several styles of composition. This happens quite naturally when a textbook is first printed on a lowcost printer, while later versions are prepared for high quality typsetting. An even more drastic case is that of transparent foils, prepared for projection, whose content is taken from a densely printed manual. The forced layout for foils suits the needs of readable projection in contrast to prints of the same material.

The advocated approach also favors the crystallization of preferred styles, typically in the form of a library of document style descriptions. For the user, such a system can naturally be tailored to his specific applications and constitutes a discreet compromise between either a too sophisticated general purpose system or a limited specialized tool.

What rules must a text obey in order to fit a given formatting style? Conversely, just what typographical variations are permissible on a given source document? The articulation is determined by the logical structure of the type of document. Let us call document description the definition of a document type in terms of its logical structure, along with the specification of the associated typographical attributes. The next section illustrates an adequate formalism for both parts of a document description.

3.2. A formalism for document descriptions

Our formalism is inspired by declarative rather than procedural methods; this should enhance the independance between logical structure description of the source document and the specification of the actual processing needed to produce the desired printout. We therefore split the document description in two tables, one for the

structure and the other for the typographical attributes.

```
Paper          = Header Abstract MainText References.
Header         = Title Authors Affiliation.
Title          = Char*.
Authors        = Char*.
Affiliation    = Char*.
Abstract       = AbstractTitle Paragraph*.
AbstractTitle  = String.
MainText       = Chapter*.
Chapter        = MainHeading Entity* Section*.
MainHeading    = String Char*.
Section        = Heading Entity* Section*.
Heading        = String Char*.
Entity         = Paragraph | TableEntity | FigureEntity.
Paragraph      = Char*.
TableEntity    = Table Caption.
FigureEntity   = Figure Caption.
Caption        = String Char*.
References     = Reference*.
Reference      = (* not specified *).
```

Figure 4: A formal grammar

The present paper is taken as an example to illustrate both parts of the formalism (figures 4, 5, 6).

3.2.1. The formal grammar as a structure description
A document is quite naturally broken into constituent parts, that occur either in a sequence or possibly as alternatives. The entire document, as well as each of its constituent elements will be called objects. An object undergoes a similar analysis and may well be decomposed itself into smaller objects. Some objects are primitive, i.e. not further amenable to decomposition; these include single characters or entities like logotypes and inserted figures.

```
Paper          :   FontFamily = ? (Times),
                   MainTextFontShape = ? (Roman),
                   MainTextFontSize = ? (10),
                   MainTextLineSpacing = ? (13),
                   MainHeadingFontShape = ? (Bold),
                   HeadingFontShape = ? (Italic),
                   CaptionFontShape = ? (Roman),
                   CaptionFontSize = ? (8),
                   CaptionLineSpacing = ? (10),
                   TextAreaWidth = ? (312),
                   TextAreaHeight = ? (528),
                   AbsoluteTopMargin = ? (100),
                   LeftMargin = ? (100),
                   RightMargin = LeftMargin + TextAreaWidth,
                   Language = English,
                   PAGINATE (AbsoluteTopMargin, TextAreaHeight);

Header         :   LeftMargin = ? (LeftMargin + 36),
                   FormattingMode = LeftAdjusted,
                   MaxVertSpace = ? (182 - 26);

Title          :   FontShape = ? (Italic),
                   FontSize = ? (14),
                   LineSpacing = ? (17),
                   TopSpace = ? (4 * LineSpacing),
                   BREAK (LeftMargin, RightMargin, FormattingMode,
                   LineSpacing);

Authors        :   FontShape = ? (Roman),
                   FontSize = ? (10),
                   LineSpacing = ? (13),
                   TopSpace = ? (2 * LineSpacing),
                   BREAK (LeftMargin, RightMargin, FormattingMode,
                   LineSpacing);

Affiliation    :   FontShape = ? (Italic),
                   FontSize = ? (8),
                   LineSpacing = ? (10),
                   TopSpace = ? (MainTextLineSpacing - LineSpacing),
                   BREAK (LeftMargin, RightMargin, FormattingMode,
                   LineSpacing);

Abstract       :   LeftMargin = ? (LeftMargin + 36),
                   FontShape = ? (Roman),
                   FontSize = ? (8),
                   LineSpacing = ? (10),
                   FormattingMode = Justified;

AbstractTitle  :   FixedWordValue =
                       CASE Language OF
                       English: 'ABSTRACT',
                       French: 'RESUME',
                       German: 'ZUSAMMENFASSUNG',
                       ELSE ? ('');

MainText       :   FontShape = MainTextFontShape,
                   FontSize = MainTextFontSize,
                   LineSpacing = MainTextLineSpacing,
                   FormattingMode = Justified;
```

Figure 5: An attribute table (first part)

```
Chapter        :  ChapterNumber = COUNTIN (Paper),

MainHeading    :  FontShape = MainHeadingFontShape,
                  BuiltString = ChapterNumber & '.',
                  BREAK (LeftMargin, RightMargin, FormattingMode,
                  LineSpacing);

Section        :  SectionNumber = COUNTIN (Chapter),

Heading        :  FontShape = HeadingFontShape,
                  BuiltString = ChapterNumber & '.' & SectionNumber & '.',
                  BREAK (LeftMargin, RightMargin, FormattingMode,
                  LineSpacing);

Paragraph      :  ParagraphNumber = COUNTIN (Section),
                  Indentation =
                        IF ParagraphNumber = 0 THEN 0,
                        ELSE em(FontSize),
                  JUSTIFY (LeftMargin, RightMargin, Indentation,
                  FormattingMode, LineSpacing, Language);

TableEntity    :  FixedWord =
                        CASE Language OF
                        English: 'Table',
                        French: 'Table',
                        German: 'Tabelle',
                        ELSE ? (''),
                  BuiltString = FixedWord & COUNTIN (Paper),
                  AlignmentMode = Centered;

FigureEntity   :  FixedWord =
                        CASE Language OF
                        English: 'Figure',
                        French: 'Figure',
                        German: 'Bild',
                        ELSE ? (''),
                  BuiltString = FixedWord & COUNTIN (Paper),
                  AlignmentMode = Centered;

Caption        :  FontShape = CaptionFontShape,
                  FontSize = CaptionFontSize,
                  LineSpacing = CaptionLineSpacing,
                  FormatingMode = Centered;
                  BREAK (LeftMargin, RightMargin, FormatingMode,
                  LineSpacing);

String         :  CHARS (BuiltString);

Table          :  SETENTITY (AlignmentMode);

Figure         :  SETENTITY (AlignmentMode);

Char           :  CHARGEN (FontFamily, FontSize, FontShape);
```

Figure 6: An attribute table (second part)

All other objects are obtained by one of the three following operations: sequential composition, alternative choice among objects ("|") and repetition ("*") of any number of some type of objects. Figure 4 shows a formal description of the type of document this paper conforms to; it is given in terms of the primitive objects

character, figure and table.

The notation used is derived from that of formal language grammars. As a consequence, some objects may be defined recursively and thus contain parts which are of the same generic type as the main objects themselves. In the example of figure 4 a section encompasses (sub-)sections in precisely this self-embedding fashion. The formal grammar that specifies a type of document is called its structure description

3.2.2. The attribute table as a complement for the formal grammar

To complete the description of a document by indicating how it should be processed, we need an additional table. An entry in the table is provided for each object that occurs in the grammar; it contains a set of attributes which are characteristic of the formatting processing for that particular kind of object. The set of attributes is not the same for every object; for instance, characters are attributed font parameters, whereas paragraphs need other data such as margins and justification mode.

In order to avoid tedious and error prone repetitions of similar attributes, an inheritance mechanism passes attributes down the hierarchy of objects. Thus FontFamily , an attribute for Paper, in the example of figure 5, is set to the actual value Times at the top level but is available to all objects throughout the description of the type Paper.

Since inheritance can be a bind, a masking mechanism is provided in compensation. Any inherited attribute may "temporarily", i.e. for the scope of a lower level object, be redefined with a different and better suited value. All hierarchically subordinate objects will inherit from that superimposed value, according to scope rules inspired by common practice in bloc-structured programming languages. For an example, consider LeftMargin as defined at the top level of Paper; its value (100, or alternately, AbsoluteLeftMargin) is redefined locally in the constituent object Header, with an indentation of 36 pica. Values of attributes are given in the notation of arithmetic expressions or symbolic constants. When preceded by a question mark the given value is considered as a default definition; the user has then the option of defining the value as he wants it to be, either within the source document, or in a separate file or even throuh an interactive dialog.

Most attributes serve as parameters for the formatting routines. Typically paragraphs are justified within given left and right margins and according to a specified formatting mode; these attributes are all inherited from the higher levels.

4. Using the system

This chapter describes how the system is used by several categories of users. The final users are in charge of the document acquisition and its preparation for printing according to a pre-established style. The skilled users have to define the document's style including the variety of its typographical appearances.

4.1. Document production by a final user

The final user of the system, as described in chapter 2, is placed in front of a multi-window screen, showing different views of the processed document; each window is tied to a formatting step.

Figure 7 illustrates the beginning sections of the present paper in the course of processing. One window is used to echo the source version of the document and another one to reflect its final aspect.

Relevant actions that may be performed on each window, from source to final document, will be specified.

4.1.1. Document editing

Necessary operations for source document manipulation include both usual editing functions and marking operations for the delimitation of typical objects in the document. To support the former it is enough to echo any typed-in keys from the keyboard and to provide a suitable basic set of primitives such as selection, cursor moves, search in plain text, and so on. To achieve the latter requires on-line recognition of markup commands, which may be given either by function keys, explicit in-line tags, or interactive mouse driven menu selection. Consistency checking must take into account the logical structure as defined by the prevailing document description.

For reasons of efficiency, it appears desirable to minimize the processing time of the high priority editing task, in order to avoid competition with other formatting processes. With this perspective, we advocate the use of a a rather simple editor, operating with a single non-proportional font.

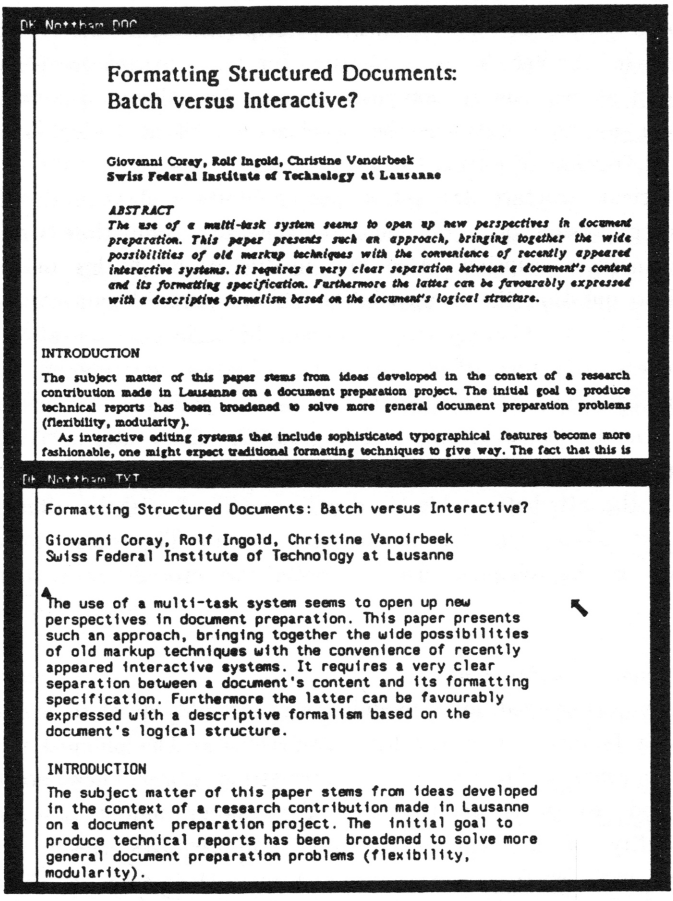
Formatting Structured Documents:
Batch versus Interactive?
Giovanni Coray, Rolf Ingold, Christine Vanoirbeek
Swiss Federal Institute of Technology at Lausanne
ABSTRACT
The use of a multi-task system seems to open up new perspectives in document preparation. This paper presents such an approach, bringing together the wide possibilities of old markup techniques with the convenience of recently appeared interactive systems. It requires a very clear separation between a document's content and its formatting specification. Furthermore the latter can be favourably expressed with a descriptive formalism based on the document's logical structure.
INTRODUCTION
The subject matter of this paper stems from ideas developed in the context of a research contribution made in Lausanne on a document preparation project. The initial goal to produce technical reports has been broadened to solve more general document preparation problems (flexibility, modularity).
As interactive editing systems that include sophisticated typographical features become more fashionable, one might expect traditional formatting techniques to give way. The fact that this is
Formatting Structured Documents: Batch versus Interactive?
Giovanni Coray, Rolf Ingold, Christine Vanoirbeek
Swiss Federal Institute of Technology at Lausanne
The use of a multi-task system seems to open up new perspectives in document preparation. This paper presents such an approach, bringing together the wide possibilities of old markup techniques with the convenience of recently appeared interactive systems. It requires a very clear separation between a document's content and its formatting specification. Furthermore the latter can be favourably expressed with a descriptive formalism based on the document's logical structure.
INTRODUCTION
The subject matter of this paper stems from ideas developed in the context of a research contribution made in Lausanne on a document preparation project. The initial goal to produce technical reports has been broadened to solve more general document preparation problems (flexibility, modularity).

Figure 7: Different views of same document

4.1.2. Document formatting

The final user is allowed to tune the formatting result by modifying the values of some typographical attributes, provided these have been set as user-definable in the document description! This can be done interactively through the window associated with the completely formatted document.

4.2. Document description by a skilled user

The specification of a document description with the formalism explained in chapter 3, is a matter for an experienced person in typography as well as in computer science formalisms. Afterwards a distinction is made between the specification of a logical structure and the definition of formatting attributes.

The logical structure description part delimits a class of documents. The source version of a document is generally compatible with only one document description. Also any modification of this description brings into question the compatibility of the source documents marked according to it. Consequently it can be relevant to aim at a completeness of formatting possibilities when conceiving a new document description. Particulary it seems judicious to provide a trivial structure construction, which gives the final user a free hand to control himself any unplanned document items.

As for the attributes table, it can be modified without altering the consistency of source documents tied to it. The most important constraint is the adequate use of formatting procedures available in library.

4.3. Formatter extensions

The system, conceived in a modular way, should allow to extend the formatting features to satisfy new requirements. The planned mean to do this consists of joining to the formatting library new procedures consacrated to new applications. This programming job is the responsibility of a computer scientist and requires clearly defined interfaces with existing packages.

Conclusion

The aimed at generality of document preparation processes can only be achieved by wide parameterization and by a modular design.

The suggested approach, originally meant to combine advantages of both batch and interactive systems, has brought to the fore two issues: the need to separate the document's content from it's presentation style, and the advantage to split the formatting process into several steps.

This decomposition is favourably achieved in a system environment, that supports time-sharing of dynamic processes.

Also, in order to suit specific user needs and to enhance

portability, we advocate the use of a declarative formalism for document type description.

While presenting a particular document description, the paper touches upon a few typographical problems (paragraph justification, page breaking, automatic numbering ...) and presents a general model to solve them. Other situations, such has foot-note handling, integration of tables and diagrams, can be settled by suitable extensions within the same framework. Among remaining open problems, the resolution of internal references, coherent with page and section numbering, introduces an additional inter-dependance among formatting processes, yet to be analysed.

A more fondamental problem raises with the enrichment of a given logical structure to accomodate more complex document processing while preserving the compatibility with existing documents.

This paper, essentially devoted to fundamental problems, hardly mentions user- dialog facilities. Convinced by the implementability of these novel concepts, we are considering an investigation oriented towards user-interfaces, as well for document production as for document description.

References

[1] J. Andre. Bibliographie Analytique sur les "manipulations de textes", in TSI, vol. 1, no 5, p. 445-455.

[2] P. Bratley, G. Coray, G. Tiphane. COMPO: un langage de description de textes, in Actes des Journées sur la Manipulation de Documents, Rennes, 4-6 mai 1983, p. 47-68.

[3] Donald D. Chamberlin, James C. King, Donald R. Schultz, Stephen J.P. Todd, Bradford W. Wade. JANUS: An Interactive System for Document Composition, in Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation, Portland, Oregon (USA), June 8-10, 1981, ACM SIGPLAN Notices, vol. 6, no 6, June 1981, p. 82-91.

[4] C.F. Goldfarb. Generalized Approach to Document Markup, in Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation, Portland, Oregon (USA), June 8-10, 1981, ACM SIGPLAN Notices, vol. 6, no 6, June 1981, p. 121-122

[5] B.W. Kernighan, L.L. Cherry. A System for Typesetting Mathematics, Communications of the ACM, vol. 18, p. 151-157, March 1975.

[6] Donald E. Knuth. TEX and Metafont: new directions in

typesetting, Digital Press, 1979.
[7] M.E. Lesk. TBL - A Program to Format Tables, in UNIX Programmer's manual, 7th edition, vol. 2, Bell Lab., Murray Hill, New Jersey, August 1978.
[8] J. Nievergelt, G. Coray, J.D. Nicoud, A.C. Shaw. Document preparation Systems, North Holland, 1982.
[9] J.F. Ossana. NROFF/TROFF User's Manual, in UNIX Programmer's manual, 7th edition, vol. 2, Bell Lab., Murray Hill, New Jersey, August 1978.
[10] B.K. Reid. The Scribe Document Specification Language and its Compiler, in Abstract of the Presented Papers, Internatinal Conference on Research and Trends in Document Preparation Systems, February 1981, p. 59-62.
[11] J. Seybold. Xerox's Star, The Seybold Report 10, 16, April 1981.
[12] A. Van Dam, J.D. Foley. Fundamentals of Interactive Computer Graphics, 1982.

Acknowledgments
This work has been partly supported by the CERS (Commission d'Encouragement à la Recherche Scientifique), grants 1220 and 1427 n.A.